2-2016

Mobile app tagging

Ning CHEN
Alibaba Group

Steven C. H. HOI
Singapore Management University, CHHOI@smu.edu.sg

Shaohua LI
Nanyang Technological University

Xiaokui XIAO
Nanyang Technological University

DOI: https://doi.org/10.1145/2835776.2835812

Mobile App Tagging*

Ning Chen⋆, Steven C. H. Hoi†, Shaohua Li‡, Xiaokui Xiao§
⋆Alibaba Group, Hang Zhou, P.R. China,
†School of Information Systems, Singapore Management University, Singapore,
‡Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly (LILY),
Nanyang Technological University, Singapore,
§School of Computer Engineering, Nanyang Technological University, Singapore
hzzjucn@gmail.com,†chhoi@smu.edu.sg,‡lishaohua@ntu.edu.sg,§xkxiao@ntu.edu.sg

ABSTRACT

Mobile app tagging aims to assign a list of keywords indicating core functionalities, main contents, key features or concepts of a mobile app. Mobile app tags can be potentially useful for app ecosystem stakeholders or other parties to improve app search, browsing, categorization, and advertising, etc. However, most mainstream app markets, e.g., Google Play, Apple App Store, etc., currently do not explicitly support such tags for apps. To address this problem, we propose a novel auto mobile app tagging framework for annotating a given mobile app automatically, which is based on a search-based annotation paradigm powered by machine learning techniques. Specifically, given a novel query app without tags, our proposed framework (i) first explores online kernel learning techniques to retrieve a set of top-N similar apps that are semantically most similar to the query app from a large app repository; and (ii) then mines the text data of both the query app and the top-N similar apps to discover the most relevant tags for annotating the query app. To evaluate the efficacy of our proposed framework, we conduct an extensive set of experiments on a large real-world dataset crawled from Google Play. The encouraging results demonstrate that our technique is effective and promising.

Keywords

Mobile app markets; app tagging; online kernel learning

1. INTRODUCTION

With the rocketing development of mobile applications (a.k.a "apps"), app market has drawn much more attention among researchers within data mining communities. App market is a new form of software repository which contains a wealth of multi-modal data associated with apps, e.g., text descriptions, screenshot images, user reviews, and so on. Such app market data is (i) large in volume; (ii) changing rapidly; and (iii) potentially useful for various stakeholders in the mobile app ecosystem. However, we discover that most mainstream app markets (e.g., Google Play, Apple App Store, Amazon Appstore, etc.) currently do not explicitly contain tags for apps. In this paper, an app tag is referred to as a keyword which indicates the core functionality, main content or key concept of an app.

App tagging can benefit different stakeholders of mobile app ecosystems. For users, app tags facilitate them browse and locate their desired apps. For developers, terse and concise tags help them elevate the possibility of their apps being discovered by users, which leads to more downloads and profits. For app platform providers, app tags are useful for various purposes, e.g., categorizing apps in a much finer granularity. App tags also help improve app search quality in practice, e.g., a recent WSDM talk [28] from Tencent MyApp (a China app market serving hundreds of millions of users) showed that mining tags can enrich query and app representations so as to improve app search engine quality. The huge potential value of app tags motivates this study.

App tagging is a nontrivial and very difficult problem. One naive way is to let humans assign tags for apps. However, it is extremely labour intensive and time-consuming to manually tag a huge number of apps (Google Play and Apple App Store have more than 1 million apps). Therefore, it has become a pressing need to develop an automatic app tagging approach. In this paper, we propose to tackle automated mobile app tagging by exploring a search-based annotation paradigm which has been proved effective for image and video tagging [22, 20, 26]. In general, our proposed search-based app tagging framework comprises two main stages. Given a novel query app without tags, the first stage goes looking for a set of apps that are semantically most similar to the query app by searching a large app database. To facilitate the similarity search process, we learn an effective app similarity function by using our proposed Averaged Adaptive Online Kernel Learning (AAOKL) algorithm which leverages multi-modal heterogeneous data in app markets. The second stage aims to automatically discover some relevant tags for the query app from the text data (e.g., "Description") of both the query app and its nearest neighbour apps. To achieve this goal, we formulate it as an automatic keywords extraction task [7], and propose an unsupervised App Tag Extraction (ATE) approach. Finally, the top-ranked tags obtained by ATE are recommended to annotate the query app.

To evaluate the efficacy of our proposed search-based app tagging framework, we conduct an extensive set of experi-

*This work was done while the first author was a PhD candidate at Nanyang Technological University, Singapore.

WSDM'16, February 22–25, 2016, San Francisco, CA, USA.

DOI: http://dx.doi.org/10.1145/2835776.2835812

ments based on a real-world dataset crawled from Google Play. The encouraging results have demonstrated that our technique is effective and promising. For example, Figure 1 presents two app tagging results achieved by our proposed framework. The first row of Figure 1 shows the logos, names and categories (in Google Play) of these two apps: *Asphalt 7: Heat* and *Piano Melody Free*. The second row of Figure 1 shows top 10 tags annotated (correct ones are in red). Supplementary materials (including additional results, datasets, etc.) are publicly available at http://apptag.stevenhoi.org/.

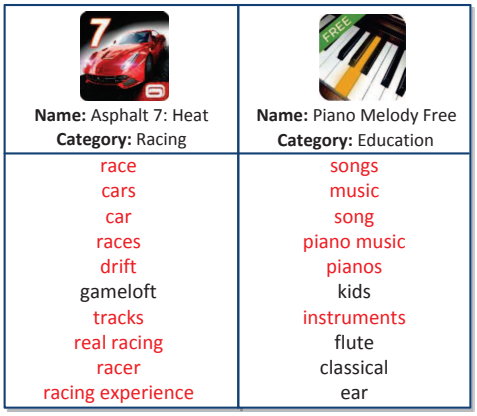

Figure 1: Two examples showing the tagging results attained by our proposed framework (AAOKL + ATE). The correct tags are highlighted in red color.

In summary, this paper makes the following contributions:

- We investigate a new research problem of automated mobile app tagging by mining emerging mobile app market data. To the best of our knowledge, this is the first comprehensive work to address this problem;
- We propose a novel search-based app tagging framework by mining multi-modal app market data, comprising two key new techniques: (i) a new online kernel learning algorithm; and (ii) a new unsupervised app tag extraction approach;
- We conduct an extensive set of experiments to evaluate the performance of our proposed auto app tagging framework on a large real-world dataset.

The rest of the paper is organized as follows: Section 2 discusses some related work; Section 3 gives the problem formulation; Section 4 introduces the proposed search-based app tagging framework; Section 5 presents the dataset used in this work; Section 6 shows the empirical results; Section 7 discusses some limitations and threats to validity; finally Section 8 concludes this paper.

2. RELATED WORK

In this section, we group related studies into three categories, and survey the literature of each category in detail.

2.1 App Markets Mining and Analysis

App markets contain a wealth of multi-modal data associated with apps, which is potentially useful for different app ecosystem stakeholders. In recent years, there are emerging studies on mining app markets data to facilitate various applications, such as app reviews mining and analysis [4], app recommendation [27], and so on. Although the nature of data used in the above studies is similar to ours, the techniques used and research goals are very different. Our work aims to use the knowledge discovered from app markets data to automatically tag apps with suitable keywords.

To the best of our knowledge, there is one previous study which is closely related to our work. Chen et al. [3] studied the problem of modeling high-level mobile app similarity and presented a framework named *SimApp* to tackle this problem. Generally speaking, our work differs from their work mainly in that we improve the *SimApp* framework and further combine it with a novel app tag extraction approach to address a different task, i.e., automatic app tagging.

2.2 Search-based Annotation

In the last few years, search-based annotation paradigm has been applied to a variety of media types, e.g., images [19, 20, 22], faces [18, 17], videos [26], and so on. For example, Wu et al. [22] proposed an online learning framework to optimize distance metric for search-based image annotation by mining a mass of social images. Zhao et al. [26] presented a data-driven video annotation framework which (i) first retrieves visual duplicates; (ii) then recommends representative tags.

While similar in spirit, our work differs in three aspects: (i) for our search-based app tagging, the feature representation of an app is quite different from that of an image (or video); (ii) our work explores a very different technique, i.e., online kernel learning, to improve the similarity search process in our framework. (iii) we propose a more advanced tag extraction approach rather than a simple majority voting to resolve our unique and challenging app tagging task.

2.3 Online Learning

We propose to learn the app similarity function using online learning techniques [9], a family of efficient and scalable machine learning algorithms, which has found promising results for many large-scale applications, such as classification and regression [8], multimedia search [23], social media, cybersecurity, and so on. Our work differs from the existing studies in that (i) we formulate and solve a challenging problem in a new application domain with distinct features; (ii) unlike traditional classification tasks, our online kernel learning aims to learn an optimal combination of multiple kernel functions from multi-modal data in an online fashion.

This work is also related to our previous study in [3], which proposed an Online Kernel Weight Learning (OKWL) algorithm that follows the first-order optimization, i.e., online gradient descent, which may suffer slow convergence. In this paper, we present a new improved online kernel learning algorithm named "AAOKL" by exploring averaged stochastic gradient descent [24] and adaptive subgradient methods [6] to improve the convergence rate significantly.

3. PROBLEM FORMULATION

This section formally formulates the problem of automatic app tagging. We first define a mobile app a_i as follows:

DEFINITION 1 (MOBILE APP). Each mobile app a_i is represented by a set of n modalities: $a_i = [m_{i1}, m_{i2}, ..., m_{in}]$, where each $m_{ij} (1 \leq j \leq n)$ denotes one modality (feature type) of mobile app a_i.

In this work, we explore the multi-modal data representation in app markets to represent an app. For example, Figure 2 shows an example of 10 modalities associated with the "YouTube" app in Google Play.

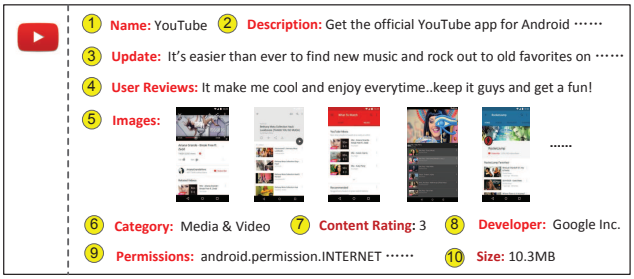

Figure 2: Example of the multi-modal information associated with the "YouTube" app in Google Play. The names of modalities are in red colour.

Definition 2 (App Tag). An app tag is a succinct and concise keyword or phrase that describes the core functionality, main content or key concept of the app.

According to Definition 2, an *app tag* should be terse and concise. Therefore, in this work, we restrict an *app tag* to be unigram or bigram ("term" in short). To better understand our definition of *app tag*, we use the tagging results shown in Figure 1 for illustration. For example, "songs", "music" and "piano music" are considered to be tags of the app *Piano Melody Free* in our problem, since they indicate the main content (and functionality) of this app, while "kids" and "flute" do not. Note that, some terms in the *Category* and *Name* attributes of an app can also be viewed as its tags, e.g., "racing" is a tag of the game *Asphalt 7: Heat*. However, in this work, we focus on automatically discover more semantic tags beyond *Category* and *Name* of apps.

Definition 3 (Automatic App Tagging). Given an app a_i, the goal of automatic app tagging is to automatically assign a list of tags to a_i by exploiting the metadata of both app a_i and a set of its semantically similar apps $\mathcal{S}(a_i)$.

In our problem, we aim to use the metadata of (i) a target app a_i; and (ii) a set of nearest neighbour apps of a_i together to discover some tags to annotate app a_i. We adopt such scheme since (1) it is often difficult to automatically find meaningful tags only relying on an app's own metadata, especially when the metadata is limited; and (2) we hope that the metadata of those nearest neighbor apps can help provide more knowledge and clues to improve the quality of the target app's tags. In this work, the metadata used for extracting app tags refers to the "Description" and "Update" (changes of the latest version) text of apps.

Remark. This work focuses on automatically discovering app tags (as defined in Definition 2) by exploiting metadata of apps in app markets. We have noticed that other kinds of tags (e.g., tags based on users' searching intentions) can be mined from other data sources (e.g., users' searching logs), which however is out of the main scope of this work.

In general, the automatic app tagging problem faces two challenges. First, given a novel app, we require an effective app similarity function to find the query app's semantically similar apps. Second, we need an effective tag extraction scheme which can automatically discover high quality app tags from a bunch of text data. In Section 4, we will present our solutions to address both challenges in detail.

4. OUR APP TAGGING FRAMEWORK

In this section, we first give an overview of our proposed app tagging framework to address the problem stated in Definition 3, and then present each component in detail.

4.1 Overview

Figure 3 presents the architecture of our proposed framework for automatic app tagging. The framework depicted in Figure 3 is essentially a search-based app tagging approach. Solid arrows in Figure 3 present the main process of the proposed framework. Specifically, when a query app (without tags) is submitted (1), we first conduct a similarity search process (2) to find N apps that are most semantically similar to the query app from a large app database (3). Then, the "Description" and "Update" text of the query app and its' top-N similar apps (4) are utilized by our proposed App Tag Extraction (ATE) approach to automatically discover a list of terms for the query app (5). Finally, the top-ranked terms are recommended to tag the query app (6). Note that, in this work, we consider the most complex case, i.e., **no** apps in our app database contain any tags, since app tagging is currently not supported by most mainstream app markets. Dashed arrows in Figure 3 present the process for learning the app similarity function. Specifically, given the stream of training data generated from a subset of the app database (I), we develop an online kernel learning algorithm to incrementally learn the app similarity function which is then adopted in the similarity search process (II).

In general, there are two major challenges for the proposed auto mobile app tagging framework as shown in Figure 3. The first is to learn an effective app similarity function efficiently to facilitate the app similarity search. We attempt to tackle it by developing a new online kernel learning algorithm by learning from multi-modal data in app markets (see Section 4.2 and 4.3). The second challenge is to automatically extract relevant tags from a bunch of text data effectively. To solve this, we propose an unsupervised app tag extraction approach (see Section 4.4).

4.2 Measuring App Similarity by Kernels

A kernel function can be viewed as a pairwise similarity function. In this work, we follow the same set of 10 kernel functions as defined in [3], denoted as $K_k (1 \leq k \leq 10)$, to measure the app similarity in different modalities as shown in Figure 2 (For more details, please refer to [3]).

4.2.1 Name

Each app has a *name*, which is essentially a short string of characters. Therefore, we adopt the well-known string kernel [11] to measure the app similarity in this modality. Let n_i and n_j denote the names of apps a_i and a_j, respectively. We can then define the kernel on the name modality as follows:

$$K_1(a_i, a_j) = \sum_{u \in \Sigma^*} \phi_u(n_i)\phi_u(n_j)$$

where Σ^* denotes the set of all subsequences, u denotes a subsequence, and ϕ is a feature mapping function. For space limitation, please refer to [11] for more technical details.

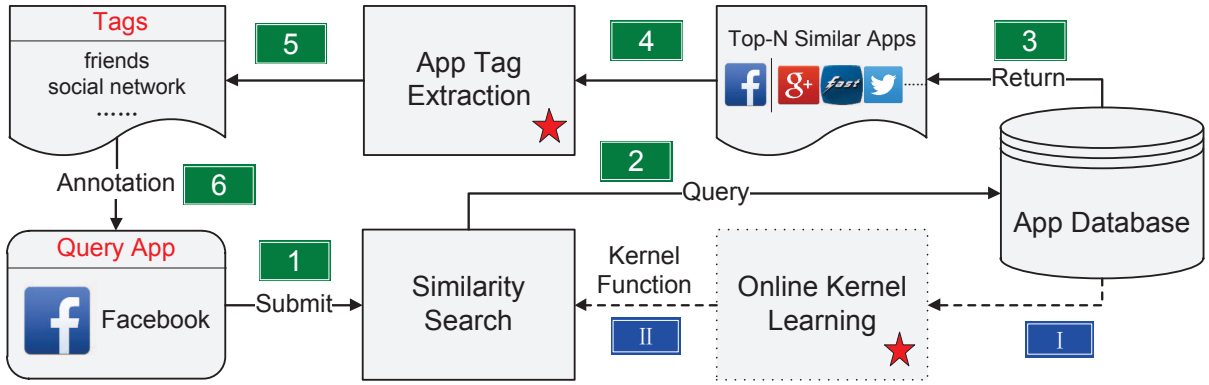

Figure 3: The system architecture of the proposed auto mobile app tagging framework. The key challenges we address in this work include (i) an online kernel learning algorithm; and (ii) an app tag extraction approach (see Figure 4 in Section 4.4).

4.2.2 *Descriptions, Updates, and User Reviews*

In this work, we exploit three types of **text** data, i.e., *descriptions*, *updates* and *user reviews*. We employ the same scheme used in [25] to measure the app similarity in these three modalities. Next, we use the text *description* modality as an example to illustrate the idea.

We collect all the apps' descriptions and treat them as documents. Then, we use LDA [1] to learn the topic distribution of each description. Thus, each app description can be represented as a fixed length vector $\mathbf{t} \in \mathbb{R}^{|T|}$, where $|T|$ denotes the number of description topics. Let $\mathbf{t}_i$ and $\mathbf{t}_j$ denote the description texts of a_i and a_j, respectively. We measure the app similarity in this modality by using the following normalized liner kernel:

$$K_2(a_i, a_j) = \frac{\mathbf{t}_i \cdot \mathbf{t}_j}{\|\mathbf{t}_i\| \, \|\mathbf{t}_j\|}$$

Following the same way, we denote $\mathbf{u}_i$ and $\mathbf{u}_j$ as the *update* text of apps a_i and a_j, respectively. The similarity between a_i and a_j in this modality is given by,

$$K_3(a_i, a_j) = \frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\| \, \|\mathbf{u}_j\|}$$

It is slightly different for *user reviews*, i.e., we need to gather all the user reviews of an app together as a document. Let $\mathbf{r}_i$ denote the review topics distribution of app a_i, we have,

$$K_4(a_i, a_j) = \frac{\mathbf{r}_i \cdot \mathbf{r}_j}{\|\mathbf{r}_i\| \, \|\mathbf{r}_j\|}$$

4.2.3 *Images*

We apply the bag-of-(visual)-words (BoW) to represent an app in visual space. We extract SIFT features from images, cluster the SIFT features using the fast K-means algorithm, then quantize the SIFT descriptors of a given image into the set of $|V|$ "visual words" (clusters), and finally compute a compact BoW representation $\mathbf{v} \in \mathbb{R}^{|V|}$. Typically, an app a_i has more than one screenshot image, so we use the centroid of all screenshots belong to a_i to represent it in the visual space, i.e., $\overline{\mathbf{v}}_i = \sum_m \mathbf{v}_m / |a_i|$, where $\mathbf{v}_m$ denotes the visual representation of an image of a_i, and $|a_i|$ denotes the total number of images belong to a_i. We measure the visual similarity between a_i and a_j using the RBF kernel as:

$$K_5(a_i, a_j) = exp\left(-\frac{\|\overline{\mathbf{v}}_i - \overline{\mathbf{v}}_j\|^2}{2\sigma^2}\right)$$

where σ is the bandwidth parameter. By default, we set σ as the average Euclidean distance in this paper.

4.2.4 *Category and Content Rating*

We explore two types of **nominal** data, i.e., *category* and *content rating*. Let c_i and c_j denote the category lables of apps a_i and a_j, respectively, we have,

$$K_6(a_i, a_j) = \begin{cases} 1 & \text{if} \quad c_i = c_j \\ 0 & \text{if} \quad c_i \neq c_j \end{cases}$$

Let cr_i and cr_j denote the content ratings of app a_i and a_j, respectively. We measure the similarity between a_i and a_j in this modality as follows,

$$K_7(a_i, a_j) = \begin{cases} \beta^{|cr_i - cr_j|} & \text{if} \quad |cr_i - cr_j| < cr_{max} - cr_{min} \\ 0 & \text{if} \quad |cr_i - cr_j| = cr_{max} - cr_{min} \end{cases}$$

where cr_{max} and cr_{min} represent the maximum and minimum ratings in the app market's content rating system, respectively. β is a decay factor within the range of [0,1].

4.2.5 *Developer*

Every app is associated with a *developer*. Each developer usually has some specialized categories of apps. First, we collect all the category labels in the app market and build a category dictionary with size $|D|$. Then, the developer of each app is represented as a vector $\mathbf{d} \in \mathbb{R}^{|D|}$ following the *tf-idf* scheme. Let $\mathbf{d}_i$ and $\mathbf{d}_j$ denote the developers of apps a_i and a_j, respectively. We measure the app similarity in this modality by employing the RBF kernel, i.e.,

$$K_8(a_i, a_j) = exp\left(-\frac{\|\mathbf{d}_i - \mathbf{d}_j\|^2}{2\sigma^2}\right)$$

4.2.6 *Permission*

We use the bag-of-words model to represent the permissions required by an app. First, we collect all the permissions and compile a dictionary with size $|P|$. Then, the permissions of an app are converted into a vector in $\mathbb{R}^{|P|}$ using the *tf-idf* weighting scheme. In this way, an app a_i can be represented as a vector $\mathbf{p}_i \in \mathbb{R}^{|P|}$. We also apply the RBF kernel to measure the app similarity in the permission space, i.e.,

$$K_9(a_i, a_j) = exp\left(-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\sigma^2}\right)$$

4.2.7 Size

Let s_i and s_j denote the size (storage space) of app a_i and a_j, respectively. We measure the similarity between a_i and a_j in this modality using the Laplacian Kernel, formally,

$$K_{10}(a_i, a_j) = exp\left(-\frac{|s_i - s_j|}{\gamma}\right)$$

where we set γ as the average absolute value of size difference (unit: MB) between two apps.

4.3 Learning Optimal Weights for Kernels

We propose to model the app similarity function f as a linear combination of multiple kernels, i.e.,

$$K(a_i, a_j; \mathbf{w}) = \sum_{k=1}^{n} w_k K_k(a_i, a_j)$$

where a_i and a_j are the i-th and j-th app, respectively. $K(a_i, a_j; \mathbf{w})$ is the target app similarity function f. K_k denotes the kernel function for the k-th modality of apps, n is the total number of kernels, and $\mathbf{w} \in \mathbb{R}^n$ is the weight vector with each w_k denotes the weight of the k-th kernel.

To learn the optimal combination weights $\mathbf{w}$, we follow the two key schemes adopted in [3], i.e., (i) online learning framework; and (ii) learning from side information of triple-wise app relationship. However, the Online Kernel Weight Learning algorithm, which follows the first order optimization framework, proposed in [3] suffers from the slow convergence issue. To address this limitation, in this work, we present a new online kernel learning algorithm by exploring (i) the adaptive subgradient method [6], which is a second order online optimization method; and (ii) averaged stochastic gradient descent [24, 15].

4.3.1 Averaged Adaptive Online Kernel Learning

In the training phase, we assume a collection of training data is given sequentially in the form of triplet [23], i.e.,

$$\mathcal{T} = \{(a_i, a_i^+, a_i^-), i = 1, ..., |\mathcal{T}|\}$$

where each triplet (a_i, a_i^+, a_i^-) indicates that app a_i is more semantically similar to app a_i^+ than a_i^-. Here, $|\mathcal{T}|$ denotes the total number of triplets in $\mathcal{T}$. Note that, such training triplets can be obtained by various ways in practice, e.g., using existing app search engines. Our goal is to learn a kernel function $K(a_i, a_j; \mathbf{w})$ such that all triplets in $\mathcal{T}$ satisfy,

$$K(a_i, a_i^+) > K(a_i, a_i^-) + \epsilon$$

where ϵ is a margin factor (we set as $+1$) to ensure a sufficiently large difference.

Algorithm 1 presents the core procedure of our proposed Averaged Adaptive Online Kernel Learning (AAOKL) algorithm. First of all, we introduce the inputs of AAOKL. $\mathcal{T}$ is a collection of training triplets given sequentially. λ is a regulization parameter, η_0 is a learning rate constant and t_0 specifies the start point (iteration) of the averaging process. Initially, we set $\mathbf{w}_1$ (non-averaged weights) and $\overline{\mathbf{w}}_1$ (averaged weights) to the vectors with all elements equal to $1/n$, where n is the number of base kernels, so each base kernel is assigned the same weight. $\mathbf{G}_0 \in \mathbb{R}^{n \times n}$ is the initial diagonal matrix which is used to store the historical gradients. The initial averaging rate u_1 is set to 1 (Line 1).

For each iteration t, we set the learning rate $\eta_t = \eta_0(1 + \lambda\eta_0 t)^{-3/4}$ [2] (Line 3). Then, for a triplet $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ received from $\mathcal{T}$, $i_t \in \{1, ..., |\mathcal{T}|\}$ (Line 4), we compute $\mathbf{s}_{i_t}^+$ and $\mathbf{s}_{i_t}^-$ (Line 5), respectively as follows:

$$\mathbf{s}_{i_t}^+ = [K_1(a_{i_t}, a_{i_t}^+), ..., K_k(a_{i_t}, a_{i_t}^+), ..., K_n(a_{i_t}, a_{i_t}^+)]^T$$
$$\mathbf{s}_{i_t}^- = [K_1(a_{i_t}, a_{i_t}^-), ..., K_k(a_{i_t}, a_{i_t}^-), ..., K_n(a_{i_t}, a_{i_t}^-)]^T$$

Algorithm 1: Averaged Adaptive Online Kernel Learning

Input: $\mathcal{T}$, λ, η_0, t_0
1 Initialize: $\mathbf{w}_1 = \overline{\mathbf{w}}_1 = 1/n$, $\mathbf{G}_0 = 0$, $u_1 = 1$
2 **for** $t = 1, 2, ..., |\mathcal{T}|$ **do**
3 Set $\eta_t = \eta_0(1 + \lambda\eta_0 t)^{-3/4}$.
4 Receive one triplet $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ from $\mathcal{T}$.
5 Compute $\mathbf{s}_{i_t}^+$ and $\mathbf{s}_{i_t}^-$.
6 Set hinge loss:
$l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) = max\{0, \epsilon - \mathbf{w}_t \cdot \mathbf{s}_{i_t}^+ + \mathbf{w}_t \cdot \mathbf{s}_{i_t}^-\}$.
7 **if** $l(\boldsymbol{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) = 0$ **then**
8 $\mathbf{g}_t = \lambda \mathbf{w}_t$
9 **end**
10 **else if** $l(\boldsymbol{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) > 0$ **then**
11 $\mathbf{g}_t = \lambda \mathbf{w}_t + \mathbf{s}_{i_t}^- - \mathbf{s}_{i_t}^+$
12 **end**
13 **for** $j = 1, 2, ..., n$ **do**
14 $\mathbf{G}_{t,jj} = \sum_{\tau=1}^{t} \mathbf{g}_{\tau,j}^2$
15 $\mathbf{w}_{t+1,j} \leftarrow \mathbf{w}_{t,j} - \frac{\eta_t}{\sqrt{\mathbf{G}_{t,jj}}} \mathbf{g}_{t,j}$
16 **end**
17 Set $u_t = 1/max\{1, t - t_0\}$.
18 Update $\overline{\mathbf{w}}_{t+1} \leftarrow \overline{\mathbf{w}}_t + u_t(\mathbf{w}_{t+1} - \overline{\mathbf{w}}_t)$
19 **end**
Output: $\overline{\mathbf{w}}_{|\mathcal{T}|+1}$

The objective function upon receiving the triplet $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ at iteration t is formulated as:

$$\mathcal{L}(\mathbf{w}_t; i_t) = \frac{\lambda}{2}||\mathbf{w}_t||^2 + l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-))$$

where $l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-))$ is the hinge loss for $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ which is defined as follows (Line 6):

$$\begin{aligned} l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) &= max\{0, \epsilon - K(a_{i_t}, a_{i_t}^+) + K(a_{i_t}, a_{i_t}^-)\} \\ &= max\{0, \epsilon - \mathbf{w}_t \cdot \mathbf{s}_{i_t}^+ + \mathbf{w}_t \cdot \mathbf{s}_{i_t}^-\} \end{aligned}$$

We consider sub-gradient of the objective $\mathcal{L}(\mathbf{w}_t; i_t)$ with respect to $\mathbf{w}_t$ (Line 7-12):

$$\mathbf{g}_t = \frac{\partial \mathcal{L}(\mathbf{w}_t; i_t)}{\partial \mathbf{w}_t} = \begin{cases} \lambda \mathbf{w}_t & l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) = 0 \\ \lambda \mathbf{w}_t + \mathbf{s}_{i_t}^- - \mathbf{s}_{i_t}^+ & l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) > 0 \end{cases}$$

Let $\mathbf{G}_t \in \mathbb{R}^{n \times n}$ denote the diagonal matrix at iteration t where the j-th diagonal element $\mathbf{G}_{t,jj}$ stores the sum of the squares of all past gradients of the j-th feature (kernel), i.e., $\sum_{\tau=1}^{t} \mathbf{g}_{\tau,j}^2$ (Line 14). Then, for the j-th feature, we update:

$$\mathbf{w}_{t+1,j} \leftarrow \mathbf{w}_{t,j} - \frac{\eta_t}{\sqrt{\mathbf{G}_{t,jj}}} \mathbf{g}_{t,j}$$

where $\frac{\eta_t}{\sqrt{\mathbf{G}_{t,jj}}}$ is the j-th feature's specific learning rate at iteration t (Line 15). In such a way, every feature has it's own learning rate that adapts to the data dynamically.

Except for normal weight updates, we start averaging weights at certain iteration t_0 (which can be tuned experimentally), the output $\overline{\mathbf{w}}_t$ is the average of all the weights

from iteration t_0 to t, formally,

$$\overline{\mathbf{w}}_t = \begin{cases} \mathbf{w}_t & t < t_0 \\ \frac{1}{t-t_0+1} \sum_{l=t_0}^{t} \mathbf{w}_l & t \geqslant t_0 \end{cases}$$

We compute this average using a recursive formula [2], i.e.,

$$\overline{\mathbf{w}}_{t+1} \leftarrow \overline{\mathbf{w}}_t + u_t(\mathbf{w}_{t+1} - \overline{\mathbf{w}}_t)$$

where $u_t = 1/max\{1, t - t_0\}$ is the averaging rate (Line 17-18). Finally, after $|\mathcal{T}|$ iterations, we output the learned weight vector $\overline{\mathbf{w}}_{|\mathcal{T}|+1}$ as the optimal combination weights.

Remark. Learning the app similarity function $K(a_i, a_j; \mathbf{w})$ is necessary, as shown in two cases: (i) when an app similarity function is **not** available in mobile app ecosystem, our technique can build one from scratch; (ii) when an existing app similarity function is **not** effective enough, we can improve it significantly (see Section 5.5 in [3]).

4.4 App Tag Extraction Approach

Given a novel app a_0 (without tags) as a query, we first apply the learned app similarity function to find a set of N nearest neighbor apps for this app via searching a large app database. The novel app a_0 and its N nearest neighbor apps $\mathcal{S}(a_0) = \{a_1, a_2, ..., a_N\}$ form an app set which we denote as $A_0 = \{a_0, a_1, a_2, ..., a_N\}$. Our objective is to automatically discover a list of tags associated with a_0 by mining the "Description" and "Update" text of all the apps in A_0 ($A_0.text$ in short). In this study, we formulate it as an automatic keywords extraction task [7]. Specifically, we first extract valid terms (i.e., n-grams, $n \leq 2$) from $A_0.text$. Then, we apply a common stopword list to remove terms which begin or end with a stopword on the list. The remaining terms are considered as *candidate terms*, from which we aim to find the most appropriate ones as app a_0's tags.

Figure 4: Overview of the App Tag Extraction approach.

To achieve this goal, we propose an unsupervised App Tag Extraction (ATE) approach. Figure 4 presents the overview of ATE, which consists of three steps. Specifically, the first step applies a list of pre-defined Part-of-Speech (POS) patterns to filter unlikely candidate terms. Then, the second step computes a TF-IDF score for each of the remaining terms. In the last step, we refine those terms with the highest TF-IDF scores by using a modified TextRank [13] algorithm. Finally, the top-ranked terms are recommended to tag the query app. Next, we present each step in detail.

4.4.1 Part-of-Speech Filtering

Syntactic properties of terms can be useful for identifying app tags. We use the Part-of-Speech (POS) info to remove unlikely candidate terms. Specifically, we first pre-define a list of POS patterns, and only those terms that match any of these patterns are kept. In this way, the number of candidate terms can be further reduced. Since nouns, verbs and adjectives are more likely to be tags, we define the POS patterns listed as follows for unigram and bigram, respectively (following the Penn Treebank tagset [12]):

- NN, JJ, NNS, VBP, VB, VBG, VBZ (unigram)
- NN NN, JJ NN, NN NNS, JJ NNS, NNS NN, VBG NN (bigram)

In particular, we use the Stanford POS tagger [16] to get the POS information of the candidate terms.

4.4.2 TF-IDF Weighting

Given a set of candidate terms for a query app a_0, it's not an easy task to identify which terms are more likely to be tags of a_0. In this study, we propose a measure to rank the possibility of terms to be app tags based on the following three intuitions: (1) a term appears more frequently in $A_0.text$ is more likely to be an app tag; (2) a very common term in the app domain is less likely to be an app tag; and (3) the terms from neighbor apps which are more similar to the query app a_0 is expected to have greater likelihood than those from neighbor apps which are less similar to a_0.

In particular, for each app, we combine its "Description" and "Update" texts together as one document. Let $T_0 = \{t_1, t_2, ..., t_i, ...\}$ denote the set of candidate terms extracted from the documents of all apps in A_0, where A_0 includes the query app a_0 and the top-N similar apps of a_0. The likelihood of a term t_i to be a tag of a_0 is defined as:

$$TFIDF(t_i) = IDF_i \sum_{j=0}^{N} TF_{ij} \times Sim(a_j, a_0) \qquad (1)$$

where TF_{ij} denotes the number of times t_i appears in the document of app a_j and $Sim(a_j, a_0)$ is the similarity score between apps a_j and a_0. IDF_i denotes the Inverse Document Frequency of term t_i obtained from a large auxiliary corpus, i.e.,

$$IDF_i = log(D/D_i)$$

where D_i is the number of documents in the corpus that contain term t_i, and D is the total document size of the corpus. By using Equation (1), we can get a ranked list (in decreasing order) of candidate terms denoted as $T_0^{'}$ for tagging the query app a_0. Note that Equation (1) captures all the three intuitions summarized earlier.

4.4.3 Tag Refinement by Modified TextRank

Given the ranked list $T_0^{'}$ (in decreasing order of TF-IDF score), we add a refinement step with the aim of pushing terms that are more likely to be tags to the top of the list. To archive our goal, we propose a modified TextRank [13] algorithm. Next, we present our idea in detail.

Let $T_0^{'(1/N)}$ denote the top $\lfloor |T_0^{'}|/N \rfloor$ ranked terms in $T_0^{'}$, where N is the number of similar apps retrieved from the app database, and $|T_0^{'}|$ is the total number of terms in $T_0^{'}$. We build an undirected graph $G = (V, E)$ from $T_0^{'(1/N)}$, where V is a set of vertices and E is a set of edges. Each node $v_i \in V$ corresponds to an unigram $u_i \in T_0^{'(1/N)}$, and an edge $e_{ij} \in E$ connects two nodes v_i and v_j. The weight w_{ij} of the edge e_{ij} is proportional to the semantic relevance between v_i (u_i) and v_j (u_j). Different from the original TextRank [13] algorithm which uses co-occurrence relation, we measure the semantic relevance between two unigrams (words) based on the word embedding technique.

Word embedding is an important technique in NLP where each word in the vocabulary is mapped to a dense, low-dimensional and real-valued vector. The embedded feature vector somewhat describe the semantic characteristics of the

corresponding word. Given a large auxiliary corpus, we can learn the vector representations of words in an unsupervised manner by utilizing the *skip-gram* architecture proposed in [14] (detailed settings will be given in Section 6.2.3). We expect such representation of words is beneficial to measuring the semantic relevance between two words.

Let $\mathbf{u}_i$ and $\mathbf{u}_j$ denote the vector representations of unigrams u_i and u_j, respectively. We measure the semantic relevance $w_{i,j}$ between u_i and u_j using a polynomial kernel:

$$w_{i,j} = \left(\frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\| \, \|\mathbf{u}_j\|} + c \right)^d$$

where $w_{i,j}$ is a non-negative value, and we set $c = 1$, $d = 2$.

The TextRank (TR) score of a node v_i (which represents u_i) is defined as follows:

$$TR(v_i) = (1-d) + d * \sum_{v_j \in (V - v_i)} \frac{w_{ij}}{\sum_{v_k \in (V-v_j)} w_{jk}} TR(v_j) \quad (2)$$

where $V - v_i$ denotes the set of all the nodes in V except v_i; and $d \in [0, 1]$ is a damping factor which we set to 0.85 (the same as in [13]) in our implementation. The larger value of $TR(v_i)$, the greater probability of u_i to be an app tag.

To get the TR scores of unigrams, initially, all nodes are assigned the same scores of 1.0. Then, Equation (2) is applied iteratively until convergence (we set threshold = 0.0001). For each bigram in $T_0^{'(1/N)}$, we set its TR score as the average value of TR scores of the two words of the bigram. After the TR scores of all the terms in $T_0^{'(1/N)}$ are obtained, for each term $t_i \in T_0^{'(1/N)}$, we define its final score as the product of TF-IDF score and the normalized TR score, formally,

$$F(t_i) = TFIDF(t_i) \times \tilde{TR}(t_i) \quad (3)$$

Finally, the top-ranked terms (in terms of final score) are selected as tags for the input query app.

5. DATASET

In this section, we introduce a real-world dataset crawled from Google Play for conducting our empirical evaluations. Table 1 presents some statistics of this dataset[1].

Global App Repository: We collected app data from Google Play at full stretch. For each app, we crawled all the metadata (associated with it) available on Google Play, including description text, permissions, user reviews, etc. This yielded a *global app repository* with 1,039,127 apps in 42 categories. The *global app repository* is used as an auxiliary repository to learn (obtain) the vector representations and IDF values of words. The *retrieval database*, *training set* and *query set* discussed below are all subsets of the *global app repository*. Note that, these subsets are mutually disjoint.

Training Set: The *training set* is used for learning the app similarity function. In particular, we randomly sample 15,000 apps from the *global app repository*, and use them to generate training triplets.

Query Set: Apps in Google Play do not have tags, making it difficult for evaluating the tagging performance of our technique. Fortunately, we found three alternative Android markets (i.e., SlideME[2], Samsung Galaxy Apps[3] and PandaApp[4]), in which some apps are tagged. Such information can be utilized to build our *query set*. Specifically, we first crawl apps' metadata and their associated tags as many as possible from these three markets, respectively. Second, for each Google Play app in our *global app repository*, we try to find its duplicates from the three markets by matching app *names* and *developers*. We use the tags associated with these duplicates as the ground truth tags for the corresponding Google Play app. Third, we manually check the ground truth tags and refine their quality by employing a list of rules. Specifically, we (i) remove tags that contain non-English characters; (ii) apply a list of carefully defined stopwords to eliminate clearly noisy tags; (iii) remove tags that contain more than two words, etc. Fourth, we select an app as a candidate query app if it contains at least four ground truth tags (after removing noisy tags). Finally, we randomly sample 1000 apps from the candidate query apps (<2000 apps) as the *query set* (see Section 7 for details.).

Retrieval Database: We randomly sampled 60,000 apps from the *global app repository* as the *retrieval database*, which is served as an app knolwedge DB for search-based tagging. Empirically such a scale is able to retrieve enough highly similar apps for most queries.

	Training	Query	Retrieval
#Apps	15,000	1,000	60,000
#Permissions	115,846	7,980	459,155
#Reviews	5,600,265	1,664,430	25,608,494
#Images	115,688	10,078	463,428

Table 1: Some statistics of the Google Play dataset used in this study. "#" denotes the number of some object.

6. EMPIRICAL EVALUATION

6.1 Evaluation Metrics

To evaluate the performance of different kernel functions for measuring app similarity, we follow the two metrics as introduced in [3], i.e., Precision@K and mean Average Precision (mAP). In addition, we adopt two metrics: AP@K and HIT@K, to evaluate the tagging accuracy as follows:

- **AP@K:** The Average Precision for top-K tags. For each query app, we compute the proportion of correct tags among the top-K ranked tags, and finally compute the average AP@K over the entire *query set*.
- **HIT@K:** For each query app in the *query set*, we follow the previous study [18] to compute the hit rate at top-K annotated results, which measures the likelihood of having at least one of the ground truth tags among the top-K results. When averaged across the entire *query set*, this yields the HIT@K measure.

6.2 Experimental Setup

6.2.1 Find App-App Relevance

Before applying our proposed AAOKL algorithm, we need to generate a set of training triplets from a set of training apps $\mathcal{A}$. To generate such triplets, for each app $a_i \in \mathcal{A}$, we need to find a list of apps belong to $\mathcal{A}$ that are somewhat relevant to a_i, denoted as $R(a_i)$.

[1]The dataset is available at http://apptag.stevenhoi.org.
[2]http://slideme.org/
[3]http://apps.samsung.com/
[4]http://android.pandaapp.com/

Google Play has a *"Similar"* feature which recommends users a list of similar apps for each app. We collected a set of p ($p = 530966$) such lists $L = \{l_1, l_2, ..., l_p\}$ from the web portal of Google Play, where $l_k (1 \leq k \leq p)$ represents the k-th list. Given two apps a_i and a_j ($a_i, a_j \in \mathcal{A}, i \neq j$), let $freq(a_i, a_j)$ denote the number of lists they both appear in. If $freq(a_i, a_j)$ exceeds a pre-defined threshold θ (which is used to reduce noise, we set it equal to 2), we consider a_j to be relevant to a_i, and add a_j into $R(a_i)$. In such a way, for each app $a_i \in \mathcal{A}$, we can obtain a list of relevant apps $R(a_i)$ and a list of irrelevant apps $\mathcal{A} - R(a_i)$.

6.2.2 Generation of Triplet Instances

After the app-app relevance of the *training set* $\mathcal{A}$ (as described in Section 5) is built, we sample the training triplets $\mathcal{T}' = \{(a_i, a_i^+, a_i^-), i = 1, ..., m\}$ as follows. First of all, we randomly sample an app a_i from $\mathcal{A}$. Then, we uniformly sample an app a_i^+ from the list of apps which are similar to a_i, i.e., $R(a_i)$. Finally, we uniformly sample an app a_i^- from the list of apps which are not similar to a_i, i.e., $\mathcal{A} - R(a_i)$. In this way, we generate a set of 50K training triplets $\mathcal{T}'$ which is used through our experiments.

6.2.3 Building Vector Representations of Words

In our experiments, we perform unsupervised learning of vector representations of words by using the skip-gram architecture of the *word2vec* tool in gensim[5]. We build the training corpus by using the "Description" and "Update" text of all the apps in the *global app repository* (excluding apps in the *query set*). The resulting vocabulary contains about 0.1 million words, and each word is associated with a 300-dimensional vector. The learned word vectors are then used to compute the semantic relevance between word pairs as described in Section 4.4.3.

6.3 Evaluation of Kernel Functions

6.3.1 Compared Methods

To evaluate the performance of our proposed AAOKL algorithm in modeling app similarity, we compared it with a variety of methods listed below:

- **Single:** three informative single kernels: K_1 (Name), K_2 (Description) and K_4 (User Reviews);
- **Uniform:** $K_1 \sim K_{10}$ are uniformly combined with each kernel function having the same weight;
- **OKWL[3]:** Online Kernel Weight Learning, an Online Gradient Descent algorithm for combining multiple kernels. We follow the same parameter settings used in [3], i.e., $\eta_0 = 0.01$ and $\lambda = 10^{-4}$;
- **AAOKL:** the proposed AAOKL algorithm for combining $K_1 \sim K_{10}$. We select the parameters using a small sample of training triplets, and set $\eta_0 = 1$, $\lambda = 10^{-4}$ and $t_0 = 10$.

Note that, since the proposed AAOKL algorithm follows the online learning framework, in this study, we did not compare it with batch learning methods, such as learning to rank or kernel alignment [5], etc. We may explore and compare more methods in future work.

[5]https://radimrehurek.com/gensim/

6.3.2 Results for Triplets Classification

In the **first** experiment, we compare AAOKL with OKWL based on a triplets classification task. First, we randomly sample a set of 5000 apps as the test set from the *retrieval database* as described in Section 5. Then, we generate a set of 20K test triplets from the test set by using the method described in Section 6.2.2. Given the training triplets $\mathcal{T}'$, we run AAOKL and OKWL and trace their test errors over the training triplets as they process during learning. Figure 5 summarizes the average results over 50 runs.

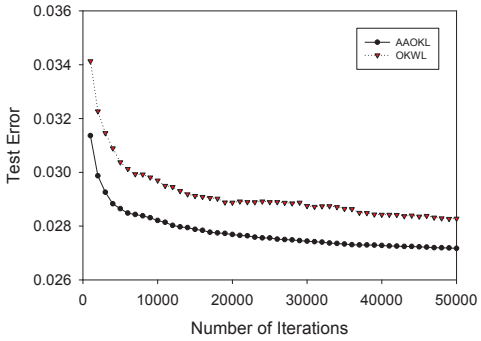

Figure 5: Test errors of AAOKL and OKWL as a function of the number of iterations (averaged over 50 runs).

We can draw some observations from Figure 5. First, AAOKL converges significantly faster than OKWL, indicating that the app similarity function learned by AAOKL can be more accurate given the same number of iterations. Second, AAOKL is more attractive since OKWL does not reach its asymptotic performance. Both observations validate that AAOKL is better than OKWL in learning app similarity.

6.3.3 Results for Similar App Recommendation

In the **second** experiment, we compare all the methods listed in Section 6.3.1 based on a similar app recommendation task. Specifically, for each query app in the test set used in the **first** experiment, we rank all other test apps according to their similarity scores to the query app, and then extract top ones as recommended apps. The ground truth (i.e., the app-app relevance of the test set) is obtained using the method described in Section 6.2.1. We measure the performance of all the compared methods in term of Precision@K and mAP. Since the largest K value in this experiment is 5, we only use test apps that have at least 5 similar apps as query apps. Table 2 shows the comparison results, from which we can draw some observations.

	mAP	K=1	K=3	K=5
AAOKL	**0.366** ±0.0007	**0.600** ±0.0012	**0.514** ±0.0010	**0.456** ±0.0012
OKWL	0.358 ±0.0013	0.592 ±0.0018	0.506 ±0.0021	0.448 ±0.0015
Uniform	0.273	0.514	0.427	0.367
Review	0.211	0.394	0.333	0.296
Desc.	0.166	0.327	0.275	0.245
Name	0.088	0.253	0.195	0.159

Table 2: mAP and Precision@K (K = 1, 3, 5) of all compared methods. The scores achieved by AAOKL and OKWL are averaged over 10 runs. "Desc." is short for "Description".

First, among the compared methods, AAOKL obtains the best results consistently in terms of both mAP and Precision@K (K=1,3,5) measures, which further validates its efficacy. Second, OKWL achieves worse mAP and Precision@K scores (with larger standard deviations) than AAOKL on this similar app recommendation task, which is consistent with the results for the triplets classification task shown in Figure 5. Third, the *Uniform* combination performs better than all the single kernels evaluated, but performs worse than AAOKL and OKWL. Such results indicate that the idea of combining different kernels is helpful, but this kind of simple strategy cannot yield the best results, thus learning the optimal combination weights of kernels in an effective way is required. Fourth, the *Review* kernel reports the best performance among all the single kernels evaluated, which indicates that user review is the most informative modality.

6.3.4 Results for App Tagging

In the **third** experiment, we fix the app tag extraction method, and evaluate the impact of app similarity functions in the search-based app tagging procedure. Specifically, for each query app in the *query set* (as described in Section 5), we apply different app similarity functions obtained (Uniform, OKWL and AAOKL) to search semantically similar apps from the *retrieval database* (as described in Section 5). In particular, a set of top $N = 10$ (the impact of varied N values will be examined in Section 6.5) similar apps are retrieved, then the proposed App Tag Extraction (ATE) approach is used to get the top-K tags for each query app. We also include "Query" as an important baseline method, which uses ATE to extract tags from the text data of the query app only, without leveraging any text data of its similar apps. The AP@K and HIT@K (K=1, 5, 10) are used as the evaluation metrics. The results are shown in Table 3, from which we can draw two observations.

Metric	Method	K=1	K=5	K=10
AP@K (%)	Query	18.70	11.78	8.58
	Uniform	22.80	13.44	10.01
	OKWL	24.70	14.76	11.12
	AAOKL	**26.60**	**15.34**	**11.33**
HIT@K (%)	Query	18.70	42.80	51.80
	Uniform	22.80	48.00	58.20
	OKWL	24.70	50.90	60.30
	AAOKL	**26.60**	**51.40**	**62.80**

Table 3: Evaluation of app similarity functions for automatic app tagging. ATE is used to extract tags. $N = 10$.

First of all, among all the methods, we found that AAOKL (+ATE) achieves the best results (in terms of both metrics) for the full range of K evaluated. For example, compared with OKWL (+ATE), AAOKL (+ATE) improves the AP@1 score by about 7.7%. This fact indicates that better learned kernel functions lead to better tagging performance.

Second, the baseline method Query (+ATE) performs much worse than the other three search-based methods in terms of both AP@K and HIT@K (K=1, 5, 10) scores. For example, AAOKL (+ATE) achieves 0.2660 while Query (+ATE) only achieves 0.1870 in terms of AP@1. Such results clearly demonstrate that (i) relying on an app's own metadata is not effective for app tagging; (ii) the effectiveness of retrieving similar apps for automatic app tagging. Due to space limitation, we give some qualitative tagging results on our website to further verify these findings.

6.4 Evaluation of Tag Extraction Methods

We also conduct an experiment to evaluate the performance of our proposed unsupervised App Tag Extraction (ATE) approach. In this experiment, we fix AAOKL as the kernel learning method and set the number of similar apps retrieved to 10 ($N = 10$). Then, three different schemes for extracting app tags are compared. Specifically, the first scheme is *TF-IDF* (the baseline), which only applies the Equation (1) (see Section 4.4.2) to rank tags. The second scheme *TF-IDF+POS* includes a Part-of-Speech (POS) filtering step (see Section 4.4.1) before applying Equation (1). Finally, ATE represents our proposed integrated approach shown in Figure 4. Table 4 summarizes the comparison results. From Table 4, we can draw two observations below.

Metric	Method	K=1	K=5	K=10
AP@K (%)	TF-IDF	23.90	14.10	10.37
	TF-IDF+POS	24.50	14.60	11.02
	ATE	**26.60**	**15.34**	**11.33**
HIT@K (%)	TF-IDF	23.90	49.60	59.60
	TF-IDF+POS	24.50	50.80	61.30
	ATE	**26.60**	**51.40**	**62.80**

Table 4: Evaluation of different tag extraction methods. AAOKL is adopted to learn the kernel function. $N = 10$.

First of all, we found that *TF-IDF+POS* performs slightly better than the baseline method *TF-IDF* in terms of both AP@K and HIT@K scores, which indicates that POS filtering is helpful in improving the tag extraction performance.

Second, with $K = 1, 5, 10$, our proposed ATE approach always achieves the best tagging performance, which shows that the tag refinement step is also be able to improve the tag extraction performance. By looking into the results, we found that, compared with *TF-IDF*, ATE improves the average precision scores by around 10%. Such fair results demonstrate the superiority of ATE in extracting app tags.

6.5 Evaluation of Varied N Values

We conduct an experiment to examine if the number of top retrieved similar apps, denoted as N, would affect the app tagging performance. Figure 6 presents the tagging performance of AAOKL+ATE (the best scheme) at top-K ($K = 1, 5, 10$) tags by varying N from 1 to 100.

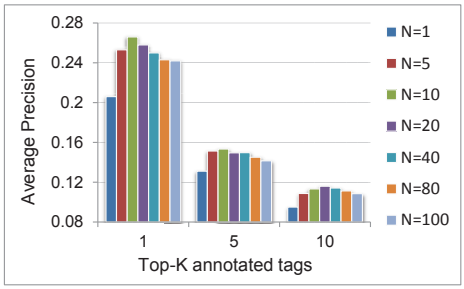

Figure 6: Comparisons of AP@K under different top-N similar apps retrieved. We apply the AAOKL + ATE scheme.

From the results shown in Figure 6, we can see that (i) when $N = 10$, AAOKL+ATE attains the best AP@1 and AP@5 scores; and (ii) when $N = 20$, AAOKL+ATE achieves

the best AP@10 score. Another observation is that, when N is smaller than 10 or larger than 40, the average precision scores drop. The reason is straightforward: if N is too small, there are not enough similar apps retrieved, while if N is too large, some less relevant apps may be retrieved, which may result in introducing noisy tags. Therefore, the app tagging performance degrades. In practice, we need to tune N empirically to achieve the best tagging performance.

7. LIMITATION & THREAT TO VALIDITY

Despite the encouraging results, this work has some limitations. First of all, the size of the *retrieval database* (60K apps) in our current experiments is relatively small for the search-based tagging process. It is relatively easy for conducting the retrieval task with this scale. However, when the *retrieval database* is very large (e.g., 1-million apps), the efficiency of similarity search can be a great challenge. In our future work, we plan to address this challenge by exploring fast approximate similarity search and indexing techniques, e.g., Kernel LSH [10]. Besides, this work also has one potential threat to validity, which relates to the generality of our app tagging framework. We validate our framework based on 1000 apps (with ground truth tags) from Google Play. It is unclear that if our technique can attain similar good or even better results when being applied to other large number of apps in both Google Play and other app markets. We may examine this issue extensively in our future studies.

8. CONCLUSION AND FUTURE WORK

This paper presented a novel auto mobile app tagging framework which aims to annotate a novel mobile app automatically by mining large amounts of multi-modal data in app markets via search-based annotation paradigms. Encouraging results from the extensive experiments validate the efficacy of our technique. Future work can (i) explore more modalities of apps; (ii) develop more effective app similarity learning techniques by other multi-modal learning [21, 23]; (iii) conduct more extensive empirical studies on other larger-scale datasets; and (iv) explore potential applications for mobile app search, browsing, categorization and beyond.

9. ACKNOWLEDGMENTS

This work was supported in part by the National Research Foundation, Singapore under its IDM Futures Funding Initiative and administered by the Interactive and Digital Media Programme Office, and Singapore Ministry of Education Academic Research Fund Tier 1 Grant (14-C220-SMU-016). This work was also supported by a grant (ARC19/14) from MOE, Singapore and a gift from Microsoft Research Asia.

10. REFERENCES

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.

[2] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. 2012.

[3] N. Chen, S. C. Hoi, S. Li, and X. Xiao. Simapp: A framework for detecting similar mobile applications by online kernel learning. In *WSDM*, pages 305–314, 2015.

[4] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. Ar-miner: Mining informative reviews for developers from mobile app marketplace. In *ICSE*, pages 767–778, 2014.

[5] C. Cortes, M. Mohri, and A. Rostamizadeh. Two-stage learning kernel algorithms. In *ICML*, pages 239–246, 2010.

[6] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[7] S. K. Hasan and V. Ng. Automatic keyphrase extraction: A survey of the state of the art. In *ACL*, pages 1262–1273, 2014.

[8] S. C. Hoi, R. Jin, P. Zhao, and T. Yang. Online multiple kernel classification. *Machine Learning*, 90(2):289–316, 2013.

[9] S. C. Hoi, J. Wang, and P. Zhao. Libol: A library for online learning algorithms. *The Journal of Machine Learning Research*, 15(1):495–499, 2014.

[10] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *CVPR*, pages 2130–2137. IEEE, 2009.

[11] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, Mar. 2002.

[12] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993.

[13] R. Mihalcea and P. Tarau. Textrank: Bringing order into texts. In *EMNLP*, pages 404–411, July 2004.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[15] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[16] K. Toutanova and C. D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *EMNLP/VLC*, pages 63–70, 2000.

[17] D. Wang, S. C. Hoi, Y. He, and J. Zhu. Mining weakly labeled web facial images for search-based face annotation. *IEEE TKDE*, 26(1):166–179, 2014.

[18] D. Wang, S. C. Hoi, Y. He, J. Zhu, T. Mei, and J. Luo. Retrieval-based face annotation by weak label regularized local coordinate coding. *TPAMI*, 36(3):550–563, 2014.

[19] X.-J. Wang, L. Zhang, F. Jing, and W.-Y. Ma. Annosearch: Image auto-annotation by search. In *CVPR*, volume 2, pages 1483–1490. IEEE, 2006.

[20] L. Wu, S. C. Hoi, R. Jin, J. Zhu, and N. Yu. Distance metric learning from uncertain side information with application to automated photo tagging. In *ACM MM*, pages 135–144, 2009.

[21] P. Wu, S. C. Hoi, H. Xia, P. Zhao, D. Wang, and C. Miao. Online multimodal deep similarity learning with application to image retrieval. In *ACM Multimedia*, pages 153–162, 2013.

[22] P. Wu, S. C.-H. Hoi, P. Zhao, and Y. He. Mining social images with distance metric learning for automated image tagging. In *WSDM*, pages 197–206, 2011.

[23] H. Xia, S. C. Hoi, R. Jin, and P. Zhao. Online multiple kernel similarity learning for visual search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(3):536–549, 2014.

[24] W. Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint arXiv:1107.2490*, 2011.

[25] P. Yin, P. Luo, W.-C. Lee, and M. Wang. App recommendation: A contest between satisfaction and temptation. In *WSDM*, pages 395–404, 2013.

[26] W.-L. Zhao, X. Wu, and C.-W. Ngo. On the annotation of web videos by efficient near-duplicate search. *IEEE TMM*, 12(5):448–461, Aug. 2010.

[27] H. Zhu, H. Xiong, Y. Ge, and E. Chen. Mobile app recommendations with security and privacy awareness. In *KDD*, pages 951–960, 2014.

[28] J. Zhuo. Semantic matching in app search. http://www.wsdm-conference.org/2015/practice-and-experience-talks/, Feb. 2015.